Automated Specification Centered Testing

Mats Heimdahl ***(PI)***

Critical Systems Research Group (CriSys), University of Minnesota

Willem Visser ***(Co-PI)***

Research Institute for Advanced Computer Science (RIACS)
NASA Ames Research Center

Sanjai Rayadurgam ***(Research Assistant)***

Mike Whalen ***(Today's Speaker)***

Critical Systems Research Group (CriSys),University of Minnesota

Automated Specification-Centered Testing

Mats P.E. Heimdahl (University of Minnesota), **Willem Visser** (NASA Ames Research Center)

Goal: Develop a specification-centered testing framework to automatically generate test cases for critical systems

Key Innovation:

- Use model-checker as test-case generation engine
- Test generation techniques independent of the software artifact (requirements, design model, code)

Challenges:

- Identify a collection of test data coverage criteria for formal, state-based specifications
- Determine a suitable translation and abstraction from a formal specification to the input language of a model checker
- Obtain concrete test-cases from abstract models
- Augment specification-based tests with test cases generated from implementation to enhance coverage

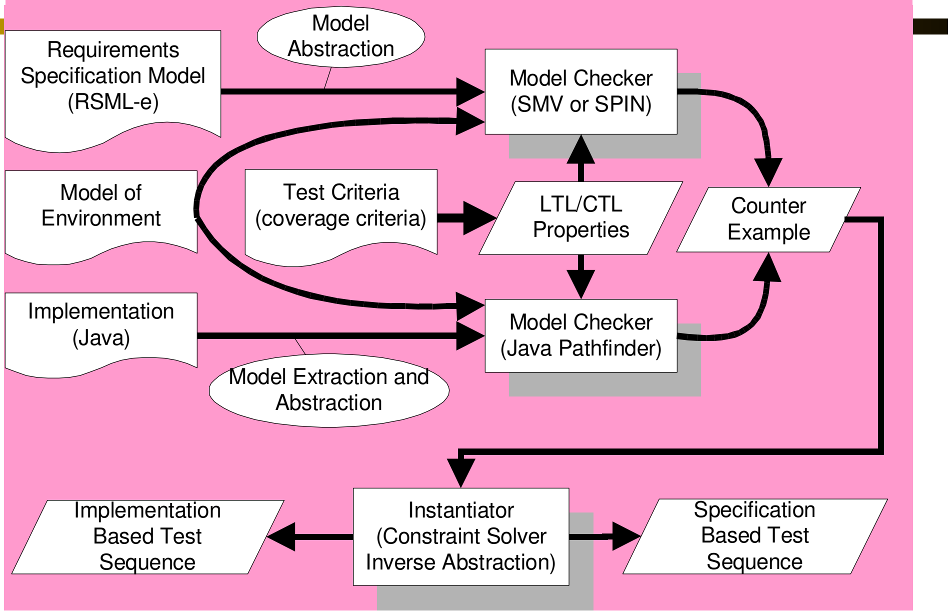
NASA Relevance:

- Increased quality and productivity of mission-critical software

Accomplishments to date:

- A framework for test-generation using model-checkers
- A set of criteria to drive test-case generation
- Domain abstraction techniques for software models
- Experiments on small models from avionics domain

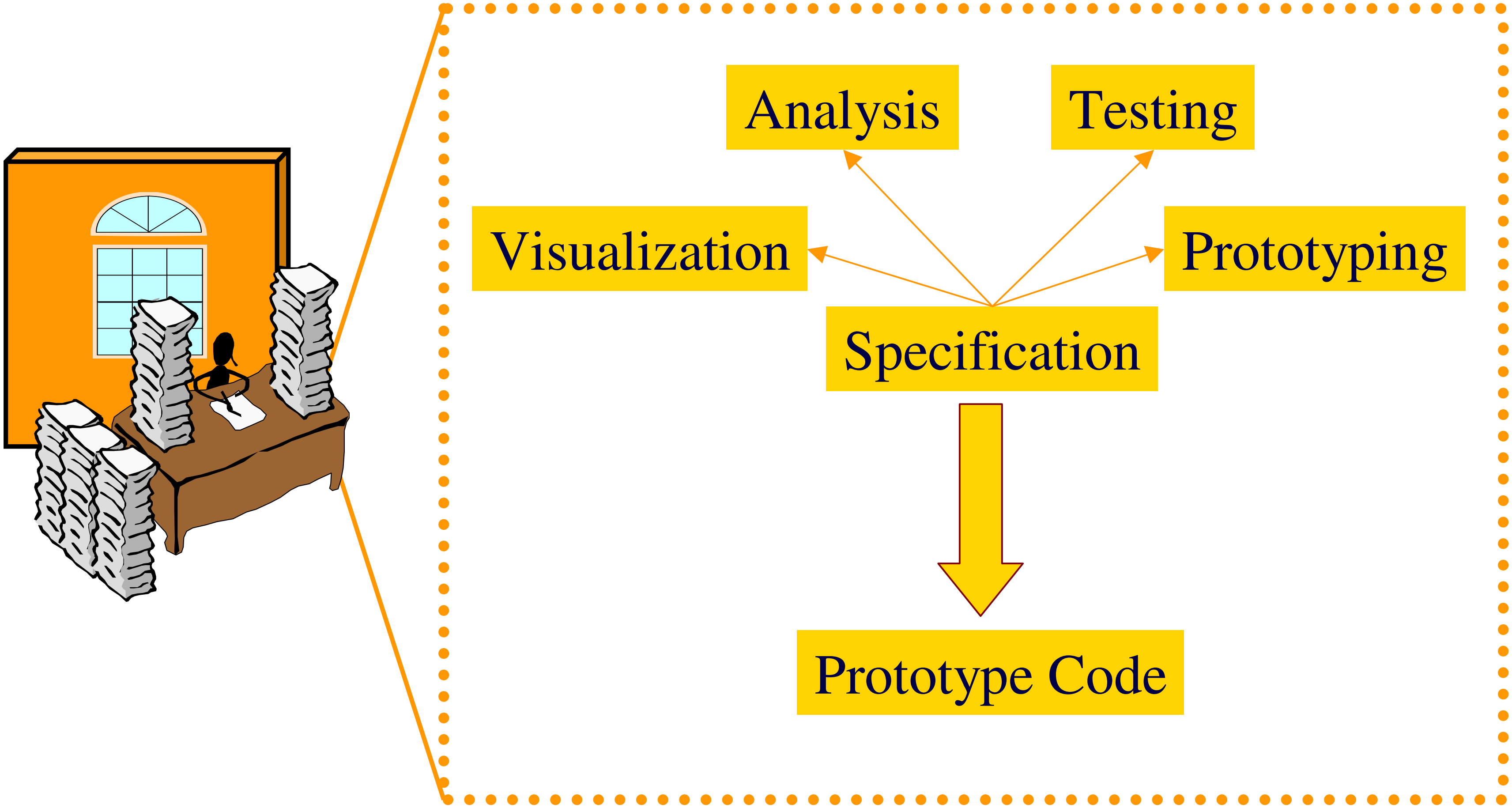
Description


Requirements Specification Model (RSML-e)
Model Abstraction
Model Checker (SMV or SPIN)
Model of Environment
Test Criteria (coverage criteria)
LTL/CTL Properties
Counter Example
Implementation (Java)
Model Checker (Java Pathfinder)
Model Extraction and Abstraction
Implementation Based Test Sequence
Instantiator (Constraint Solver Inverse Abstraction)
Specification Based Test Sequence

Schedule:

- Investigate applicability to larger requirement models of flight control logic (Fall 2002)
- Java Pathfinder model-checker enhancements for test-generation from code (Summer-Fall 2002)
- Approaches to instantiate test-cases with concrete data (Fall-Spring 2003)
- Minimizing test-suite size for a given set of criteria (Spring-Summer 2003)
- Evaluation and Wrap-up (Fall 2003)

Control Systems Workbench



Analysis

Testing

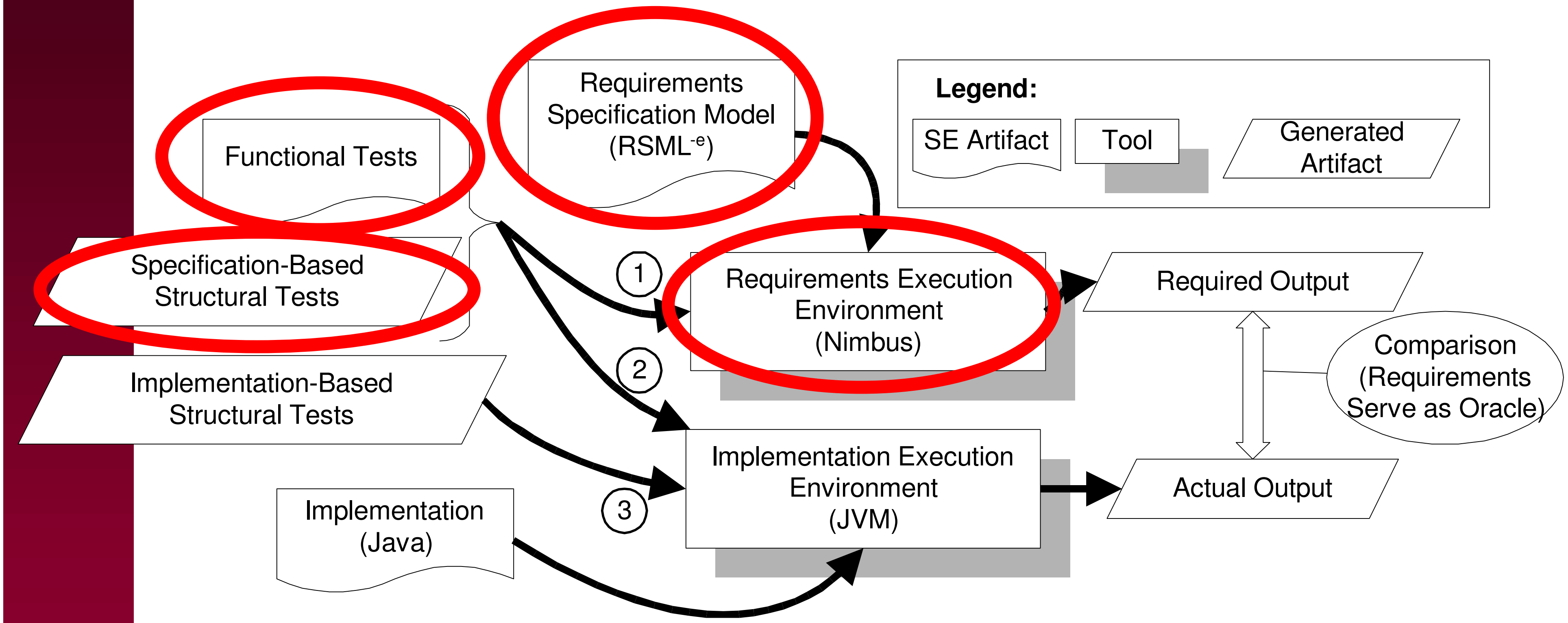
Visualization

Prototyping

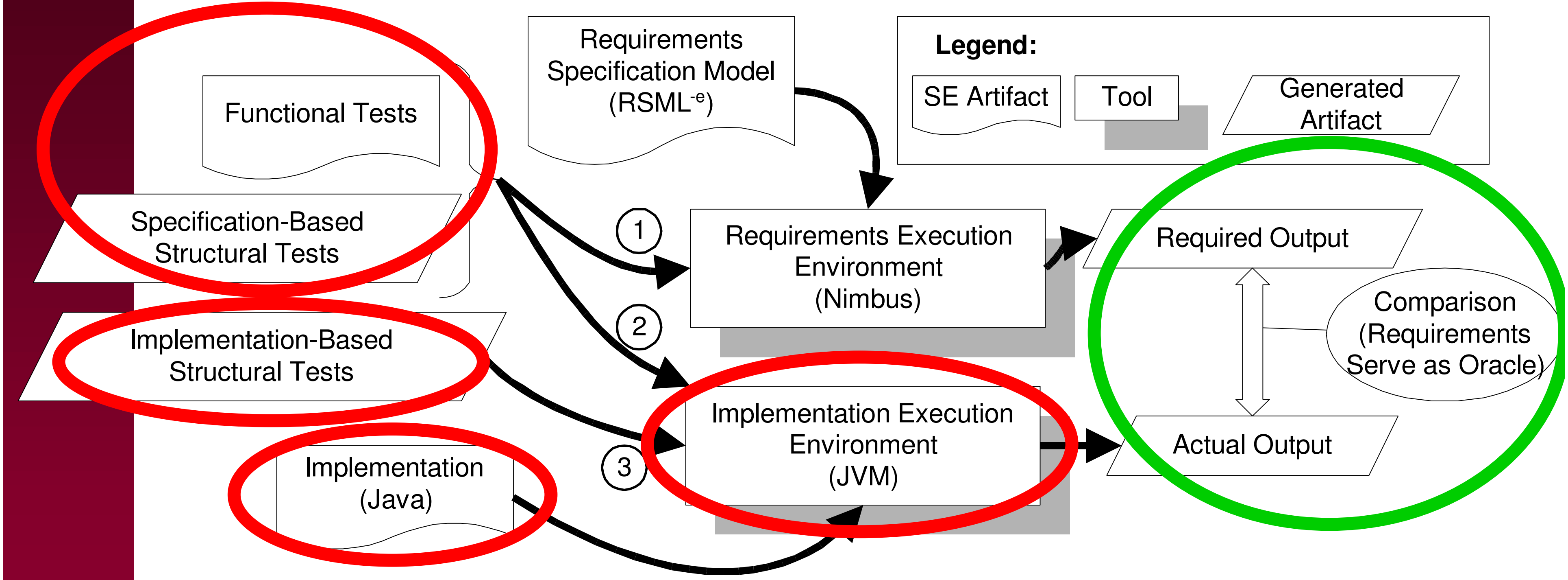
Specification

Prototype Code

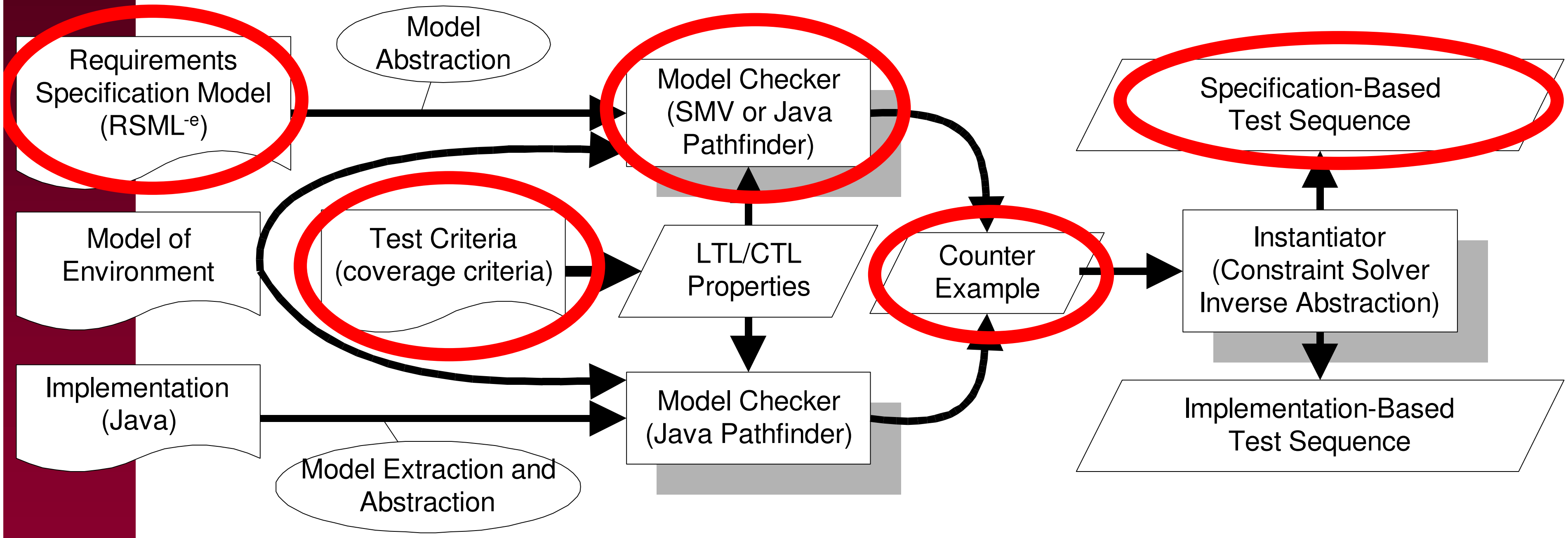
Specification Centered Testing


Legend:
SE Artifact
Tool
Generated Artifact
Functional Tests
Requirements Specification Model (RSML^{-e})
Specification-Based Structural Tests
Implementation-Based Structural Tests
Implementation (Java)
1
2
3
Requirements Execution Environment (Nimbus)
Implementation Execution Environment (JVM)
Required Output
Actual Output
Comparison (Requirements Serve as Oracle)

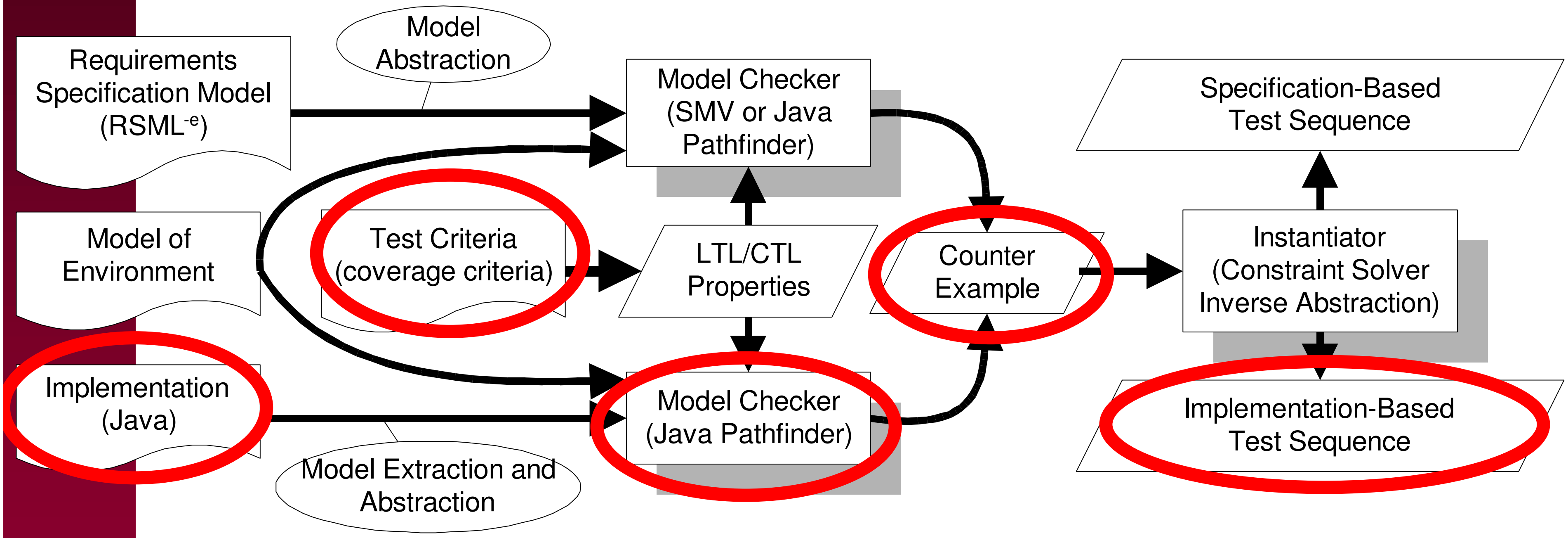
Specification Centered Testing


Functional Tests
Requirements Specification Model (RSML^{-e})
Legend:
SE Artifact
Tool
Generated Artifact
Specification-Based Structural Tests
1
Requirements Execution Environment (Nimbus)
Required Output
2
Comparison (Requirements Serve as Oracle)
Implementation-Based Structural Tests
Implementation Execution Environment (JVM)
3
Actual Output
Implementation (Java)

Generating the Tests


Requirements Specification Model (RSML^{-e})
Model Abstraction
Model Checker (SMV or Java Pathfinder)
Specification-Based Test Sequence
Model of Environment
Test Criteria (coverage criteria)
LTL/CTL Properties
Counter Example
Instantiator (Constraint Solver Inverse Abstraction)
Implementation (Java)
Model Checker (Java Pathfinder)
Implementation-Based Test Sequence
Model Extraction and Abstraction

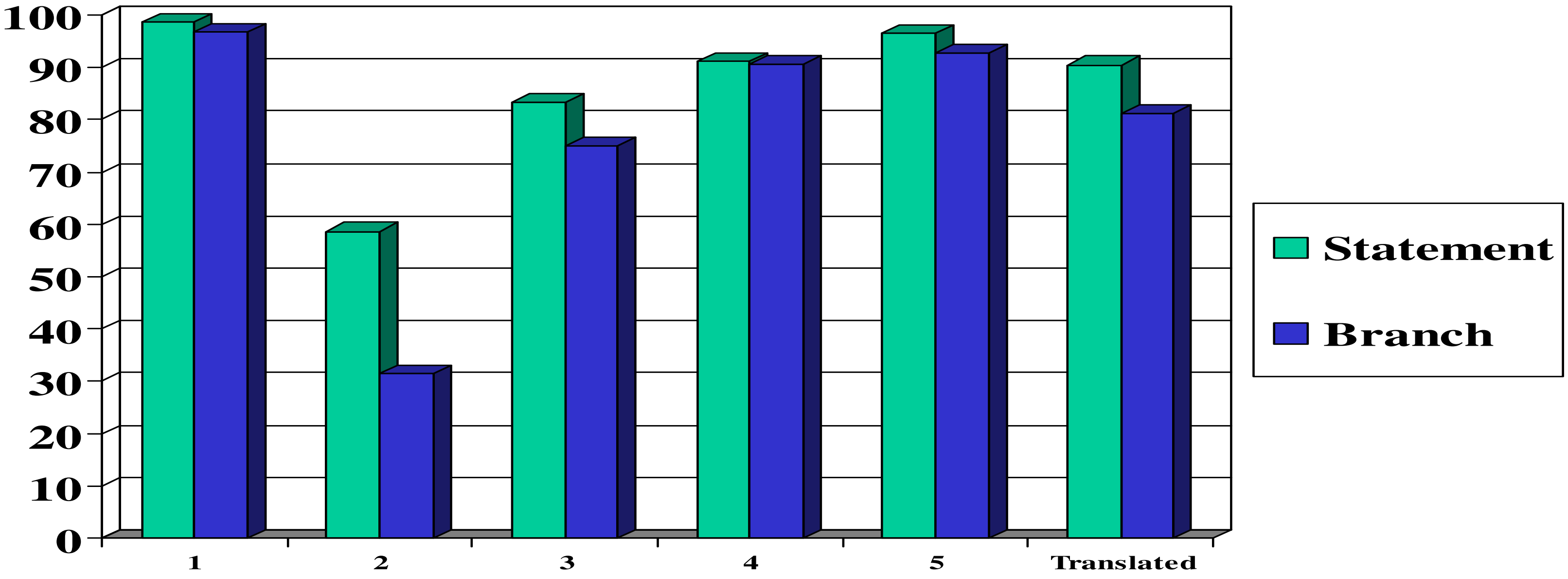
Generating the Tests


Model Abstraction
Requirements Specification Model (RSML-e)
Model Checker (SMV or Java Pathfinder)
Specification-Based Test Sequence
Model of Environment
Test Criteria (coverage criteria)
LTL/CTL Properties
Counter Example
Instantiator (Constraint Solver Inverse Abstraction)
Implementation (Java)
Model Checker (Java Pathfinder)
Implementation-Based Test Sequence
Model Extraction and Abstraction

Progress …

- Past year …
  - Formalism for generating test-cases from state-based specifications [ECBS 01/HASE 01]
  - Test criteria in terms of temporal logic [ECBS 01]
  - Framework for specification-centered testing [ICSE ATV 01]
  - Domain abstraction for software specifications [FSE 01]
- Ongoing …
  - Case-studies to investigate
    - Coverage obtained on implementations by test-cases obtained from specification
    - Scalability of the approach to large models
  - Test cases from code using JPF

Some preliminary results



- A case-study on a small avionics model (ASW)
  - RSML^{-e} specification translated to NuSMV
  - Condition coverage criteria expressed in LTL
  - Generated 49 test cases (out of 78 properties)
  - Varying coverage of implementations in Java (produced by students)
  - Coverage metrics obtained using a Java coverage tool JCover*

** Thanks to Man-Machine Systems for providing the tool free for our research project*

Issues to be tackled…

- Two critical questions
  - Can we do this on large models?
    - Approach being evaluated on models of flight-control logic from Rockwell Collins
  - Can we do this on code?
    - JPF is being used currently to generate test-cases from code obtained by translating RSML^{-e} models
  - Results from first case-studies indicate a positive answer to both.
- Other issues of interest:
  - Instrumenting Nimbus to support coverage metrics
  - Instantiating test-cases with concrete data values
  - Environment specification for generating *realistic* test cases
  - Determining pass/fail status for test cases
  - Minimizing test suite size